Programming Assignment 3 – Report

Amos Neculau, Lukas Meier

Preprocessing

Find positive and negative examples

Separating positive and negative examples for the relations happens based on the ratings given by the judges. If raters' judgement diverge, we use the dominant answer, in case of a tie we rate the relation as negative. Both datasets are stored in separate JSON files.

Resolving the IDs

We used the Google Knowledge Graph API to resolve the IDs from the Google relation corpus. The resolved objects and subjects are then stored in a JSON file so data only need to be downloaded once. As the Google's relation corpus is not fully compliant with the JSON standard, we opted to use the Python's `demjson` module, which provides more robust methods to deal with erroneous JSON files.

Features

We use the same features for both classifiers. We have used numeric features and boolean features (which we also turned into numbers: true → 1, false → -1). The following features were used:

URL in text: Feature that describes whether the last bit of the URL (the name of the Wikipedia page) can be found in the text. If the page of the name is more than one word long, the occurrence of any of these words would suffice for this feature to be true.

Subject in text: For this feature, we checked whether the resolved subject of the relation can be found in the text snippets.

Object in text: For this feature, we checked whether the resolved object name can be partially or in full be found in the text snippet. We assume that negative examples are more likely not to contain the object in their snippets.

Number of sentences mentioning the relation subject / number of sentences mentioning the relation object: These two features can give a better estimation about the existence of a relationship in the text snippets. The higher both numbers, the higher a judge would conclude that there is a relationship between subject and object.

Number of sentences: This feature simply returns the number of sentences in the text snippet.

Subject and object in direct relationship: This features indicates whether subject and object are in a direct relationship. This direct relationship is defined as "subject and object appear in

the same sentence. If this is the case this could of course be a strong indicator for that relation to exist.
Check snippet sentence for a given window: This features indicates whether the relation between the subject and the object is close. We use the number of words between subject and object as proxy. If the subject and object appear within 30 words window, this is considered "close".
Shortest path: This returns the shortest path between subject and object. This path is represented as the the number of nodes in the dependency parse tree between the object and subject. For parsing, the Stanford NLP library is used. Information on how to run it can be found in the code.
To store the features and labels, we use the ARFF format which can be read by the Weka machine learning tool. This has the advantage that it is easier to play around with the data than with Scikit learn. It is possible to run try different classifiers, remove attributes easily or display all kinds plots that allow for a better understanding of the data.

Results

Our baseline is the majority class. For both place of birth and institutions the majority of relations are positive. For the institutions, the baseline is hence at 70.69%, and at 74.051% for the place of birth relations.
As can be seen from the appendix, our institution classifier beats the baseline with an accuracy of 74.9%, while the place of birth classifier does not with an accuracy of only 73.385%. This fact is, however, also down to chance. Had the positive and negative relations in the place of birth dataset been distributed differently, we could have beaten the majority baseline also in this case.
As we did not implement the Perceptron algorithm ourselves, we decided to evaluate this algorithm using the Weka machine learning tool. The results are very similar: The accuracy is slightly worse for the institutions (74.699%) and slightly better for the place of birth relations (73.8758%).

Discussion and error analysis

The results show that some more work would needed in order for our approach to be useful. Our aim was to reach an accuracy of about 80% which we did not. On the other hand we also have to say that this task has been the first hands-on experience with machine learning for both of us. As can be seen from the results, we should have used different features for both kinds of relations as at least the feature we use perform worse on the place of birth relations than the baseline.
It is interesting to take a closer look at the features and how much they contribute to the accuracy. We did this using Weka. We found that we actually implemented too many features or should have found more features that are more telling. The "Select attribute" function of Weka reveals more about good and not so good features: We run a method on the features and their

labels which ranks attributes in terms of information gain that could be achieved using that feature. This suggests that the URL in text feature is by far outperforming all other features. If we had only used this feature, the accuracy for the institution relations would already have been around 74.7%. Basically, the additional feature make no difference which is our bad considering how much time we spent crafting them.

Appendix 1: confusion matrices for both classifiers

Institutions

Fold 1 - 75.888% accuracy

2242	334
542	515

Fold 2 - 75.53% accuracy

2214	352
537	530

Fold 3 - 74.979% accuracy

2231	341
568	493

Fold 4 - 74.374% accuracy

2185	371
560	517

Fold 5 - 74.291% accuracy

2213	342
592	486

Fold 6 - 75.557% accuracy

2248	353

535	497

Fold 7 - 75.145% accuracy

2215	361
542	515

Fold 8 - 73.988% accuracy

2175	386
559	513

Fold 9 - 75.089% accuracy

2188	362
543	540

Fold 10 - 74.154% accuracy

2190	381
558	504

Overall accuracy: 74.9%

Place of birth

Fold 1 - 73.626% accuracy

663	7
233	7

Fold 2 - 73.956% accuracy

668	6
231	5

Fold 3 - 75.275% accuracy

676	8
217	9

Fold 4 - 72.747% accuracy

652	11
237	10

Fold 5 - 72.527% accuracy

655	10
240	5

Fold 6 - 73.187% accuracy

655	12
232	11

Fold 7 - 75.055% accuracy

667	9
218	16

Fold 8 - 70.549% accuracy

637	8
260	5

Fold 9 - 72.637% accuracy

654	9
240	7

Fold 10 - 74.286% accuracy

668	7
227	8

Overall accuracy: 73.385%

Appendix 2: technical notes

The additional modules used in our scripts can all be installed through pip.
Our code is separated in two Python scripts. `ProgrammingAssingmentThree.py` would extract the features and write them (including labels) to a ARFF file. This script also includes the functionality to query the Google knowledge graph API and create JSON files in which the relations are resolved. Please note that extracting the features can take several hours. Please also note that the dependency parse tree feature is based on the Stanford CoreNLP that is accessed through a Python wrapper. Instructions on how to set Stanford CoreNLP up for our use case can be found in the code.
`Logistic_regression.py` would perform the machine learning part. It reads the data from the ARFF file, runs the scikit-learn logistic classifier on them and displays the confusion matrix including accuracy for each fold. At the end, the average accuracy is displayed. To run it, the path and name of the respective ARFF file must be added as a parameter.
All the resources we used and created, can be found in the relevant_resources subfolder of our submission.